PERFORMANCE

Dan Aronson

Introduction

The purpose of this talk to to discuss ways to get better performance out of the Connection Machine. I assume a basic understanding of PARIS and the Connection Machine architecture. Most of the ideas presented in this talk are language independant, although I will present some of these in a specific language. Some of the ideas might necessitate programming at or below paris.

Introduction (con't.)

Many of the ideas contained in this talk may apply only to the CM2 and not to future TMC products.

This is a living presentation. If any of you have performance enhancing ideas that I do not cover in this talk, please send them to performance-talk@think.com. Thinking Machines is free to do anything with messages sent to that address.

Speeds

- All numbers scaled to 65,536 processors
  - Peak
    - 32 bit weitek

      2 ops/cycle * 8Meg cycles/sec * (65,536 / 32) procs

      32.768 GFLOP

      2 ops/cycle * 7Meg cycles/sec * (65,536 / 32) procs

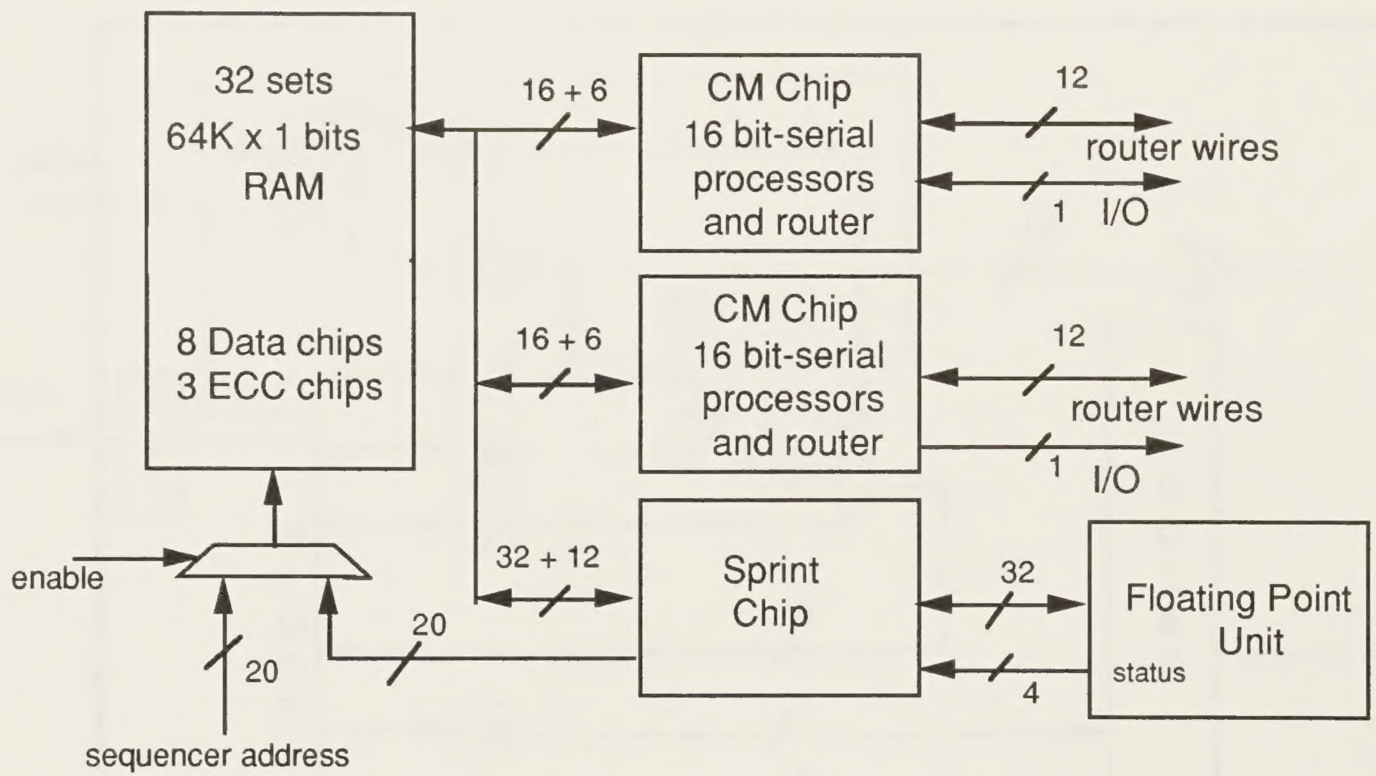
      28.672 GFLOP
    - 64 bit weitek takes two cycles to load/store 64 bit floating point numbers because of 32 bit bandwidth

Bill

Connection

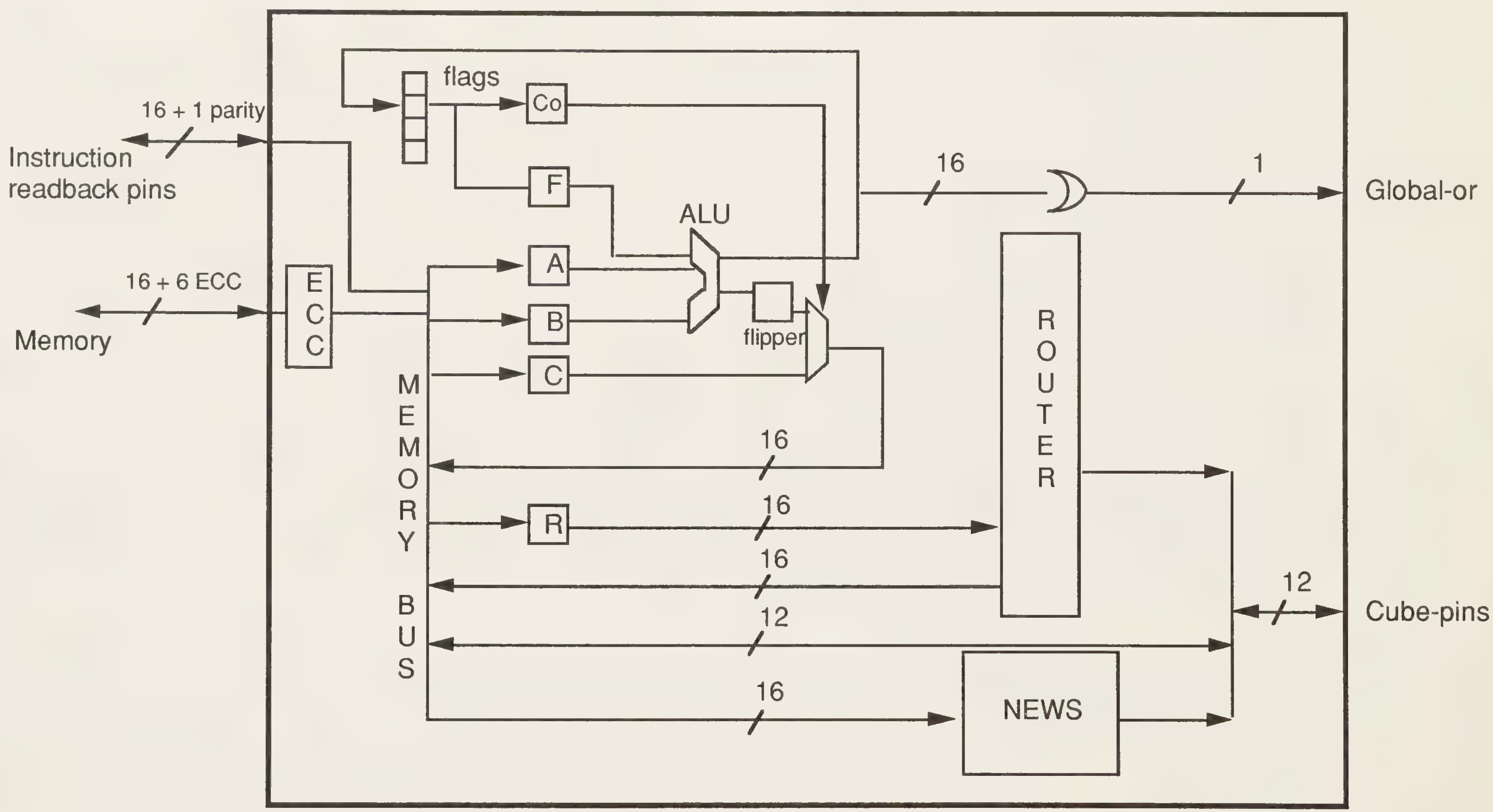
Gather/Scatter ratio

Processor Section

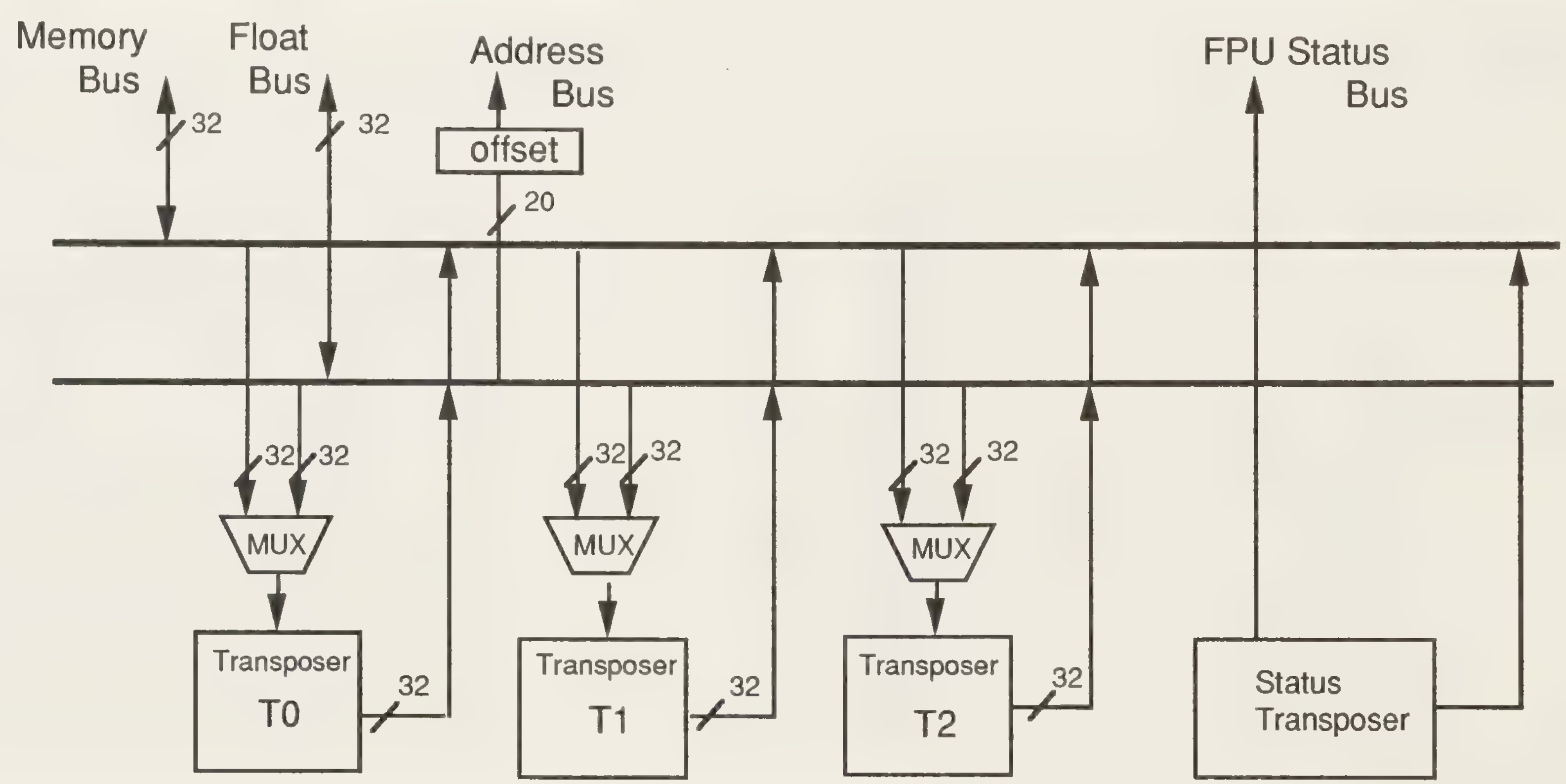


- Section is repeated 2,048 times in a full CM-2
- Instructions are broadcast
- Memory addresses can be broadcast or locally generated (indirect addressing)
- The memory and floating point units are commercial parts
- CM and Sprint chips are standard cell/custom CMOS parts

Connection Machine Chip



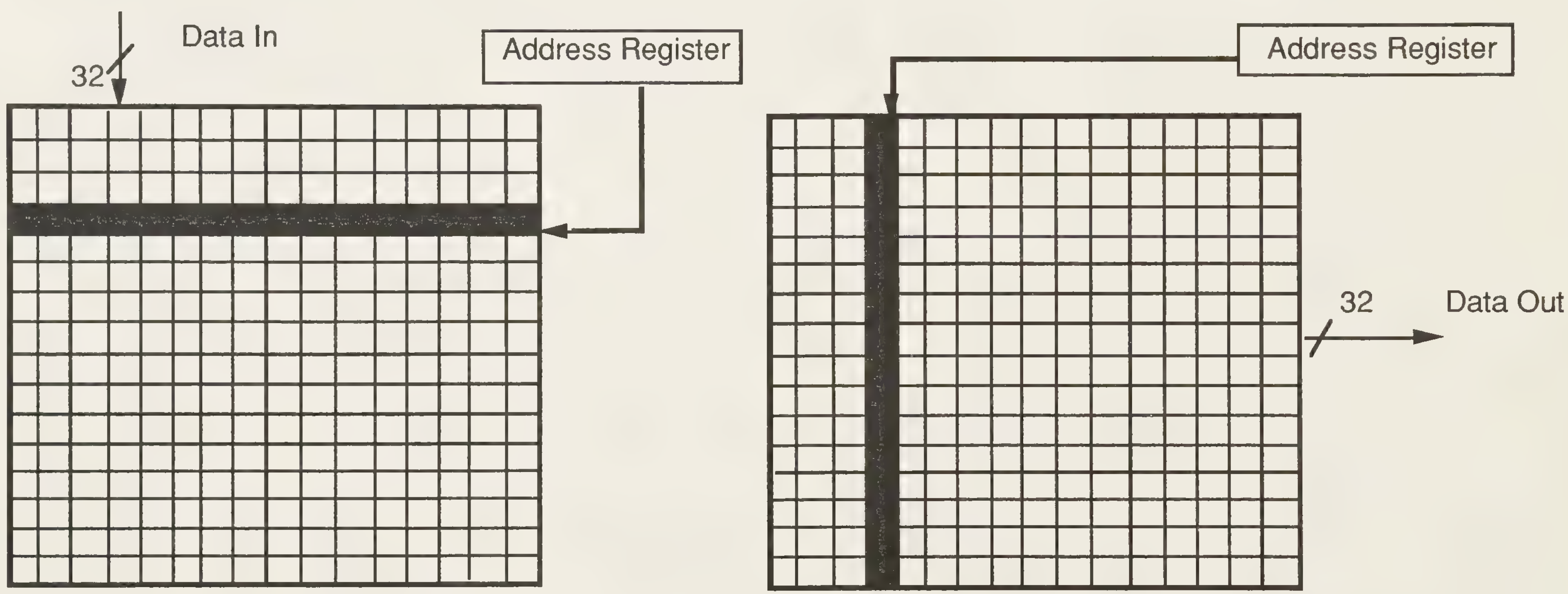
Sprint Chip



- Bit serial ⟷ word parallel conversion
- Floating point interface
- Indirect addressing

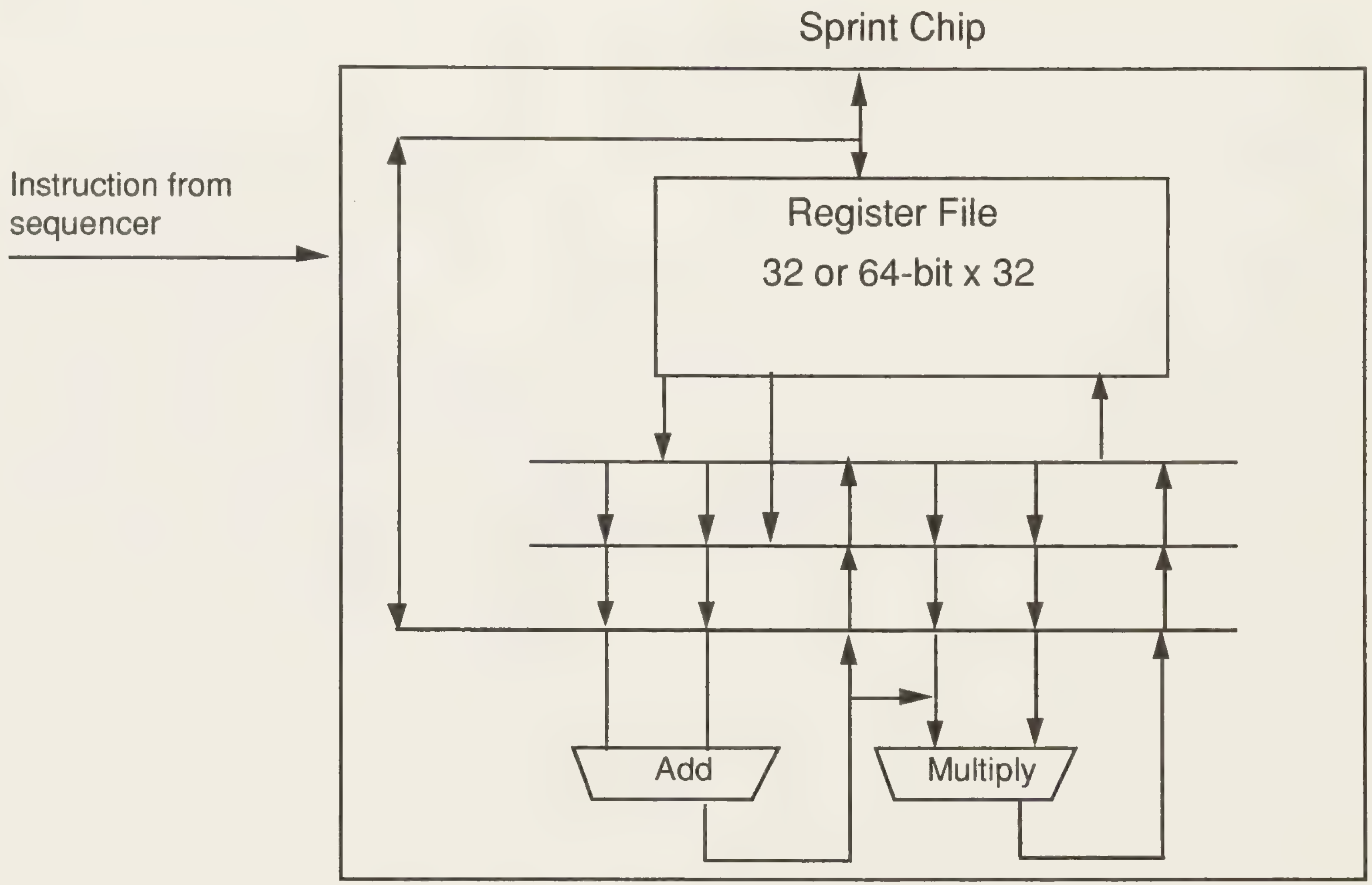
Picture

Transposer Diagram



- bit serial → • word parallel conversion
- word parallel → • bit serial conversion
- 32-cycle latency

Floating Point Unit



- Standard IEEE part
- Batch 32 processors gets vector performance
- Virtual Processor pipelining
- Floating point add or multiply is one "Add times"
- Floating point divide is 3 "Add times"

VP Ratio

- Definition

  PP - Number of Physical Processors

  VP - Number of Virtual Processors

VPR = VP/PP

Memory

- Memory
  - VPR things per physical processor
  - Can be wasteful

  Ex:

```
add_const(dest,const,length)

temp = allocate(length)
move_const(temp,const,length)
add(dest,temp,length)
```

VPR copies of constant in each physical processor

Performance

- Naive

  $C1 + VPR(C2 + TB)$

  C1: instruction overhead

  C2: vp loop overhead

  T: time per bit

  B: number of bits

  - Many important sublinear cases

Instruction Performance

Top = Tfd + Tfp + Tcm + Tother

- Tfd = Field decoding time
  - Sun4,VAX 6220 ~1µs
  - LispM, VAX 8250 ~8µs
- Tfp = Fifo Push time
  - Sun4 ~1-2µs
  - VAX ~ 2-4µs
  - LispM ~8-9 µs
- Tcm = Instruction time on the front end
  depends on VP ratio & instruction
- Tother = other front end time

VP Looping

- VP looping in microcode
  - because the cm is loosly coupled to the front end (through fifo) Tfd, Tfp and Tother can be overlapped with Tcm from previous instructions
- VP looping on front end
  - Tfd, Tfp and Tother can be overlapped with Tcm from previous instructions or previous VP loop.

Paris

- linear
  means that the formula on the CM is C1 + VPR(C2 + TB)
- integer
- boolean
- software floating point

Router

- speeding up the router
  - At higher vp ratios (>= 4), the constant overhead per iteration (C2) is higher than at lower vp ratios.
  - if possible, only route within vp banks

    at high vp ratios (>= 4) the CM using sprint routing which uses indirect addressing.

- For routing at these vp ratios, try to first move your data around within the processor, then using a bunch of low vp sends and then moving the data around within the processor again.

Router

- sometime jumbling the data speeds things up (if you have a nasty collision pattern)
- longer messages are cheaper (per bit) than shorter messages

- Sparsity

Router Compiler

Although the router is one of the key performance wins in the CM, it can be nonoptimal for static routing patterns (once that stay constant for many calls to the router).

Reason for this include:

- Dynamic redirection of messages
- Inclusion of destination address in message

Router Compiler

We have built a routing compiler that can write over the hypercube wires directly (using CMIS) and can often get a 2-3 times performance improvement over SEND's

By trying to minimize the total communication cost (number of messages over number of wiress, total hamming distance of all messages, etc) by rearranging the mapping of grid points to processors it may be possible to get another factor of 2 over that.

Paris

- Sublinear

  means that the formula on the cm is:

  C1+ VPR1(C2 + T1B)
  + VPR2(C3 + T2B + T3B)
  + VPR3(C4 + T4B)

  where VPR = VPR1+ VPR2 + VPR3,
  T2B and T3B can be overlapped

  in general (for high vp ratio's):
  VPR2 >> VPR1 and VPR2 >> VPR3

Paris

- hardware floating point
  - Transposing and Operating can be overlapped.

Examples

```
A op B -> A        32 bit :conditional or always
#cycles    MB          FB
32             B0->Td    free
32             A0->Ta    Td->Reg
32    +----->B1->Td    Ta op Reg->Reg
32    |        A1->Tb    Reg->Tc or Ta->Tc
32    |        Tc->A0    Td->Reg
32    |        B2->Tc    Tb op Reg->Reg
32    |        A2->Ta    Reg->Td or Tb->Td
32    +----->Td->A1    Tc->Reg
```

Examples

```
A op B -> C     32 bit :always
#cycles       MB          FB
32                B0->Td   free
32                A0->Ta   Td->Reg
32    +------>B1->Td   Ta op Reg->Reg
32    |           A1->Tb   Reg->Tc or Ta->Tc
32    |           Tc->C0   Td->Reg
32    |           B2->Tc   Tb op Reg->Reg
32    |           A2->Ta   Reg->Td or Tb->Td
32    +------>Td->C1   Tc->Reg
```

Examples

- compound (with constant) operation

A op Const-> B 32 bit :always

#cycles		MB	FB
2		Const->Bypass	free
1		free	Bypass->Reg[31]
32		A0->Ta	free
32		free	Ta op Reg[31]->Reg
32	+---->	A1->Tb	Reg->Tc or Ta->Tc
1	|	free	Bypass->Reg[31]
32	|	Tc->B0	Tb op Reg[31]->Reg
32	|	A2->Ta	Reg->Td or Tb->Td
1	|	free	Bypass->Reg[31]
32	+----	Td->B2	Ta op Reg[31]->Reg

Examples

Third order polynomial evaluation for all of the instructions.

Poy (A) ->A
   32 bit :conditional or :always

Comment: Constants for various functions have already been loaded into the Weitek chip. They are in temp registers

Examples

```
#cycles     MB            FB
32              A0->Ta     free
32              A1->Tb     (Ta * C3)  + C2->Reg
32        +---->free       (Ta * Reg) + C1->Reg
32        |       free     (Ta * Reg) + C0->Reg
32        |      free      Reg ->Tc
32        |      Tc ->A0   (Tb * C3)  + C2->Reg
32        |     A2->Ta     (Tb * Reg) + C1->Reg
32        |     free       (Tb * Reg) + C0->Reg
32        |     free       Reg ->Td
32        +-----Td ->A0    (Ta * C3)  + C2->Re
```

News

- 3 parts
    - on processor
    - on chip
    - off chip
    - Two measures of news cost:

News Performance

```
total-vp-ratio * (c0*length + c1)
 + (total-vp-ratio/axis-vp-ratio) * (c2*length + c3)
 + ((2^4)/(2^on-chip-bits) * total-vp-ratio/axis-vp-ratio)
    * length * c4
 + c5
```

Where the C terms are various overhead costs.

Scans

Although scans scale up sublinearly with vp ratio, it is important to remember that they run in logarithmic time in the number of physical processors in the machine.

As one researcher at TMC put it:

"Try to think about scans when you can't figure out any other way to parallize elize code".

Ex: Solving a recurrence relationship.

Solve for X(n) given the formula:

$X(i) = Z(i)^* (y(i) - x(i-1))$

$X(0) = c$

Scans

The solution can be derived by writing out several terms and examining the pattern:

$x1 = z1\ y1 - z1\ x0$

$x2 = z2\ y2 - z2\ z1\ y1 + z2\ z1\ x0$

$x3 = z3\ y3 - z3\ z2\ y2 + z3\ z2\ z1\ y1 + z3\ z2\ z1\ x0$

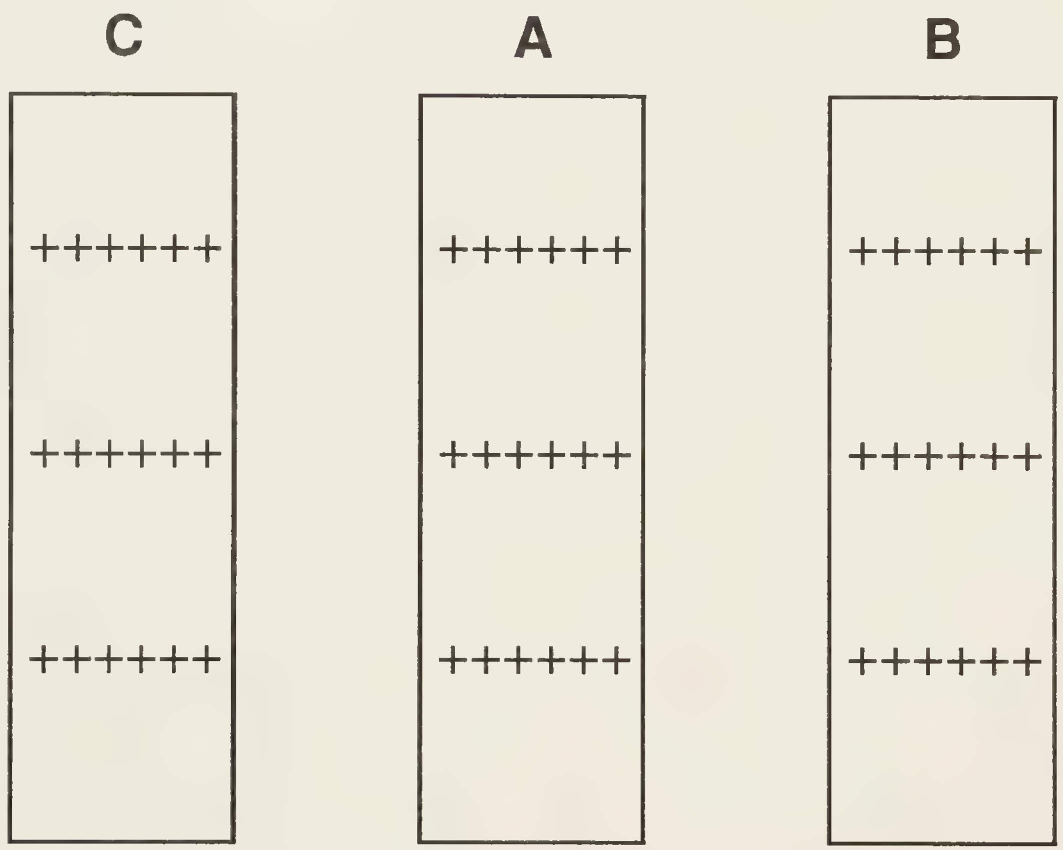
$x4 = z4\ y4 - z4\ z3\ y3 + z4\ z3\ z2\ y2 - z4\ z3\ z2\ z1\ y1 + z4\ z3\ z2\ z1\ x0$

Noticing the pattern of Z terms it becomes clear that you can solve for X(n) with a couple of elemental operations and a couple of scans.

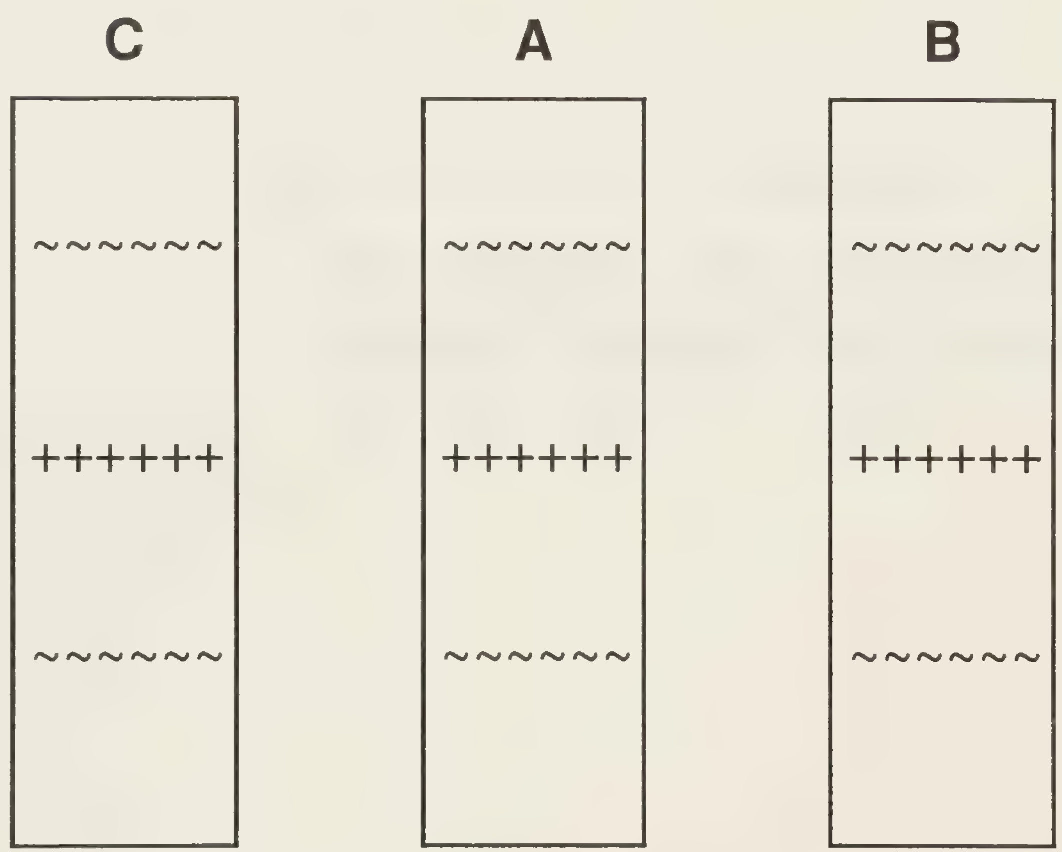
Field Aliasing

- the ability to look at memory in a different way (to slice of VP's differently)
- saving memory on temps

Field Aliasing



Field Aliasing



Aliasing

- Applicable to All Languages
  - explicit: can add as a library call
  - implicit: can have the compiler emit this

Aliasing

- increasing performance

  Remembering the formula for most paris operations with linear speedups: C1 + VPR(C2 + TB), we can examine the costs of doing a single bit logical op (always) at various vp ratios.

Aliasing

Problem: Do a 1 bit logical operation at a VP ratio of 256:

At a vp ratio of 256,

$$t = C1 + 256(C2 + 1T) = C1 + 256C2 + 256T$$

using field aliases you can opt to view the field as a 256 bit field at a vp ratio of 1,

$$t = C1 + 1(C2 + 256T) = C1 + \quad C2 + 256T$$

Obviously, if posssible using aliasing here is a big win!

Aliasing

- Can work for any logical operation
- Can work for things like unsigned add, by allowing extra overflow bits
- Caveats
  - Only works for "always" operations
  - could substantially slow things down at low vp ratios (%20)

Taking out VP looping

- many of the same advantages as field aliasing
- possibility of keeping things in registers
  - on sequencer
  - in cm chip
  - in sprint chip
  - in floating point chip
- compilers can often do this

Performance

- Other performance enhancers
  - Use "always" instructions if possible
  - Try to do conditionalization by weights
  - be careful of page faults
  - use high level primitives
  - stencils
  - math library
  - perhaps investigate CMIS

Conclusion

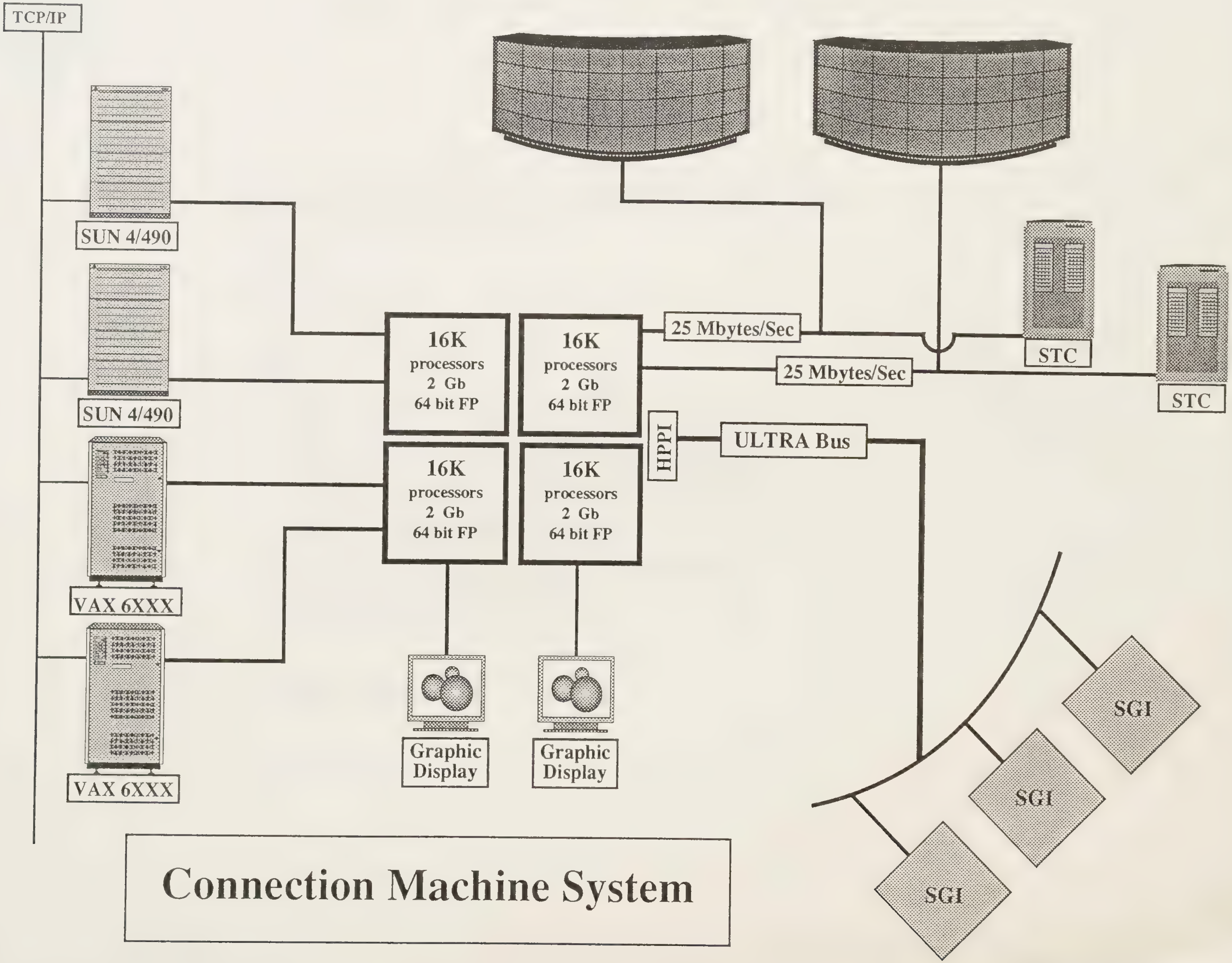
Most CM applications only get a fraction of the achievable FLOP/MIP rate. At TMC we are working on improving the languages and compilers to get closer to peak. In the interim there are a lot of various ways to get more performance (some much easier than others). By using some of these means (with or without help from TMC), it is often possible to get substantial code speed ups.

Acknowlegements

Creon Levit	(NASA)	creon@orville.nas.nasa.gov
Dan Aronson	(TMC)	dan@think.com
Guy Blelloch	(CMU, TMC)	guyb@sam.cs.cmu.edu
Mark Bromley	(TMC)	bromley@think.com
Denny Dahl	(TMC)	denny@think.com
Brewster Kahle	(TMC)	kahle@think.com
JP Massar	(TMC)	massar@think.com
Bernie Murray	(TMC)	bernie@think.com
Alex Vasilevsky	(TMC)	alex@think.com

Connection Machine Architecture

Dan Aronson

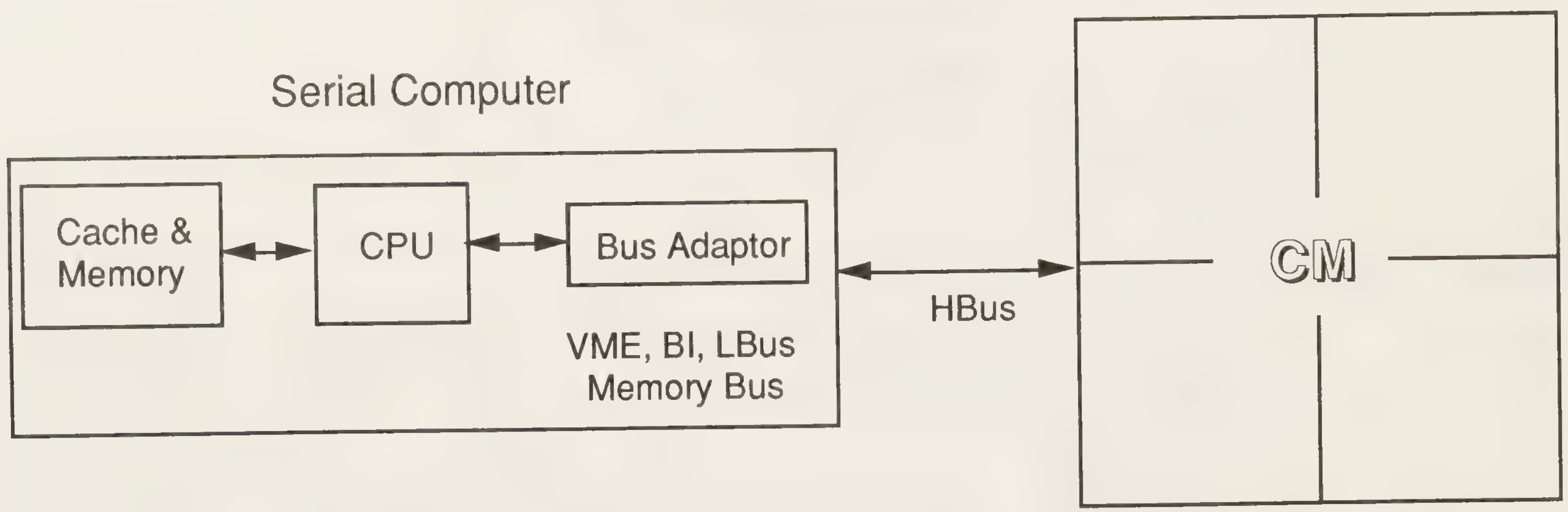


Connection Machine System

OUTLINE

- Connection Machine system architecture
  - Front end
  - CPU
    - Bit Serial Processors
    - Grid Communication and Scans
    - Router (combining, backwards, indirect addressing)
    - Indirect addressing hardware
    - Floating Point Unit
  - Frame buffer
  - DataVault

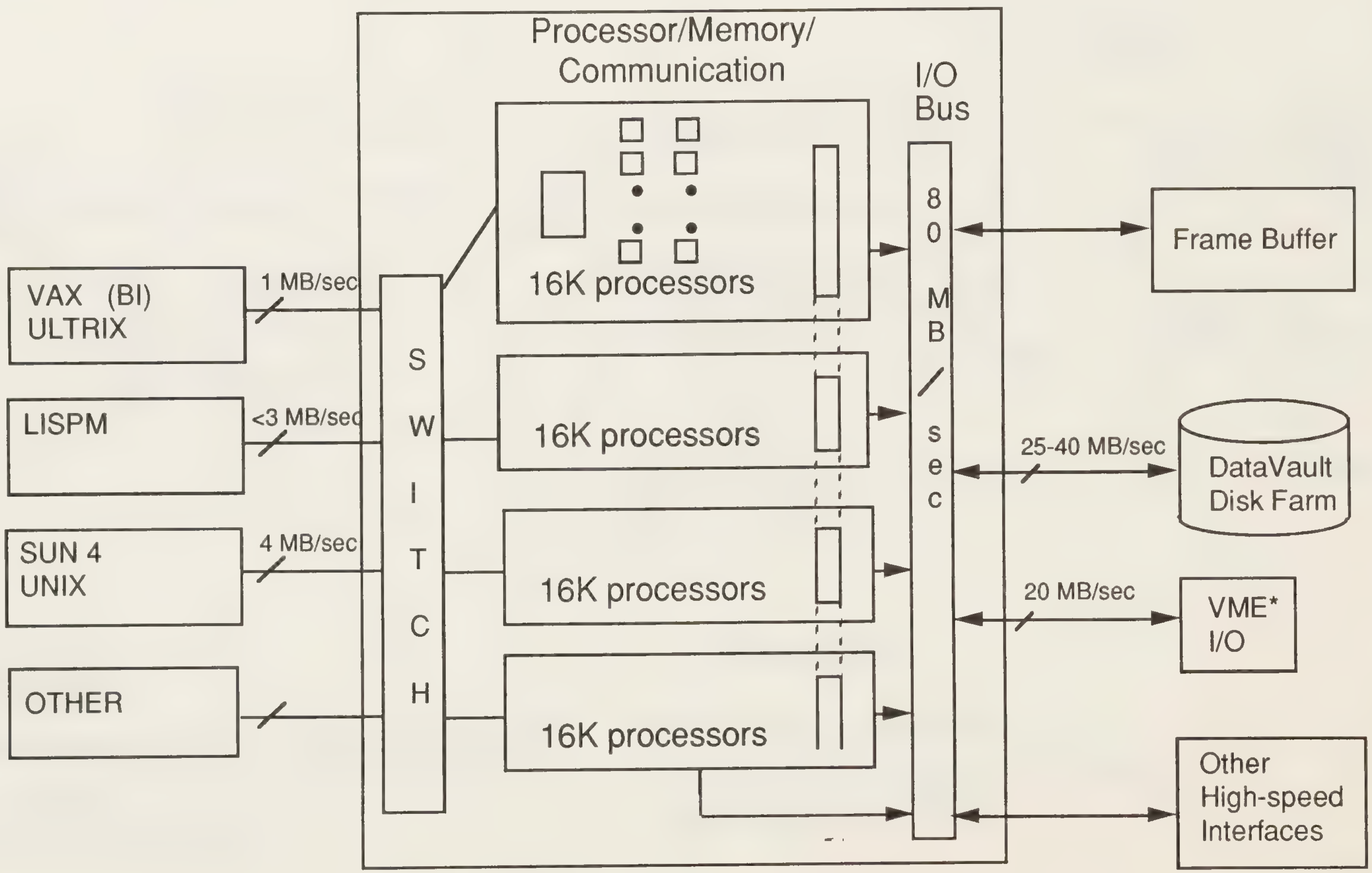
Front End Architecture



- User programs run on front end
- Macrocode calls are made to the sequencer
- Front end and CM sequencer are closely coupled
- Performance for array transfer:

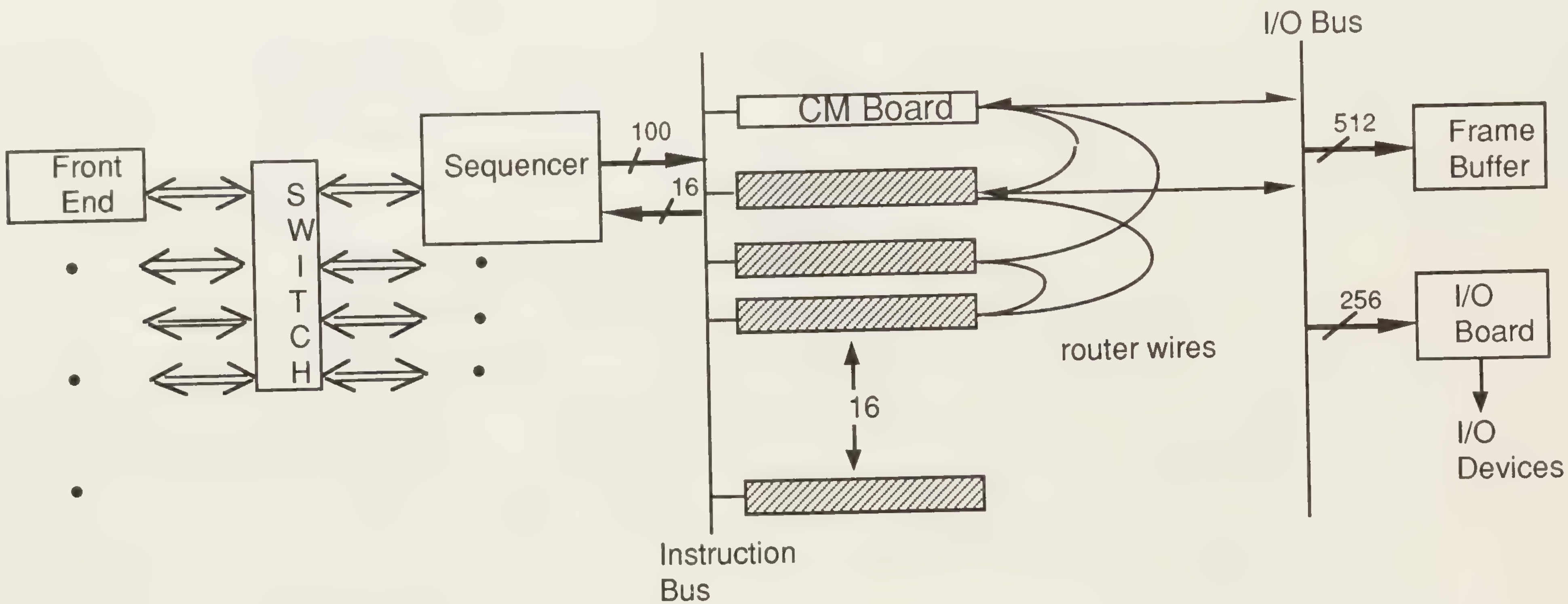
  - VAX 8800 ⟷ sequencer 1 MByte per sec
  - LISPM ⟷ sequencer < 3 MBytes per sec
  - SUN 4 ⟷ sequencer 4 MBytes per sec

The Connection Machine System



* not completed

16K Processor CPU



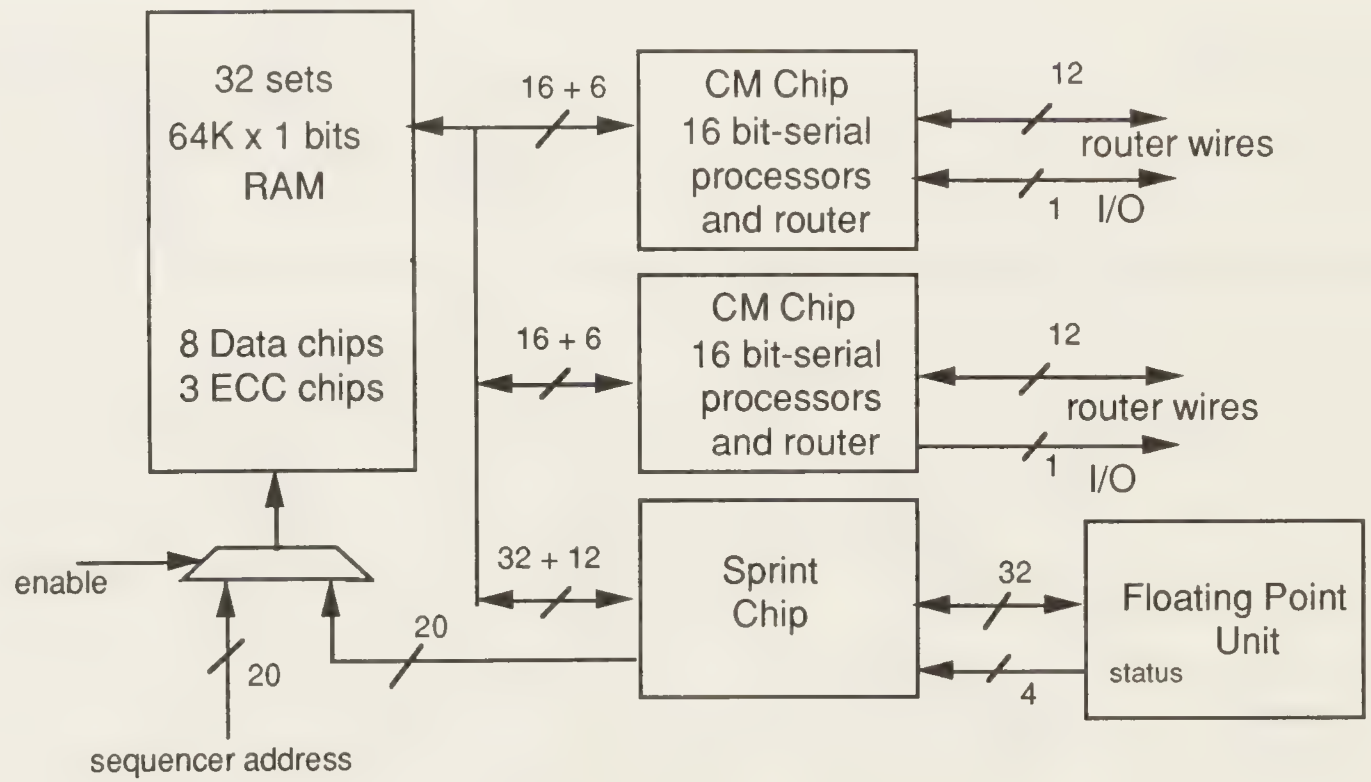
<u>Switch (Nexus)</u>

- Connects front ends to 16K processor blocks
- Full cross bar
- Broadcast instructions

<u>Sequencer</u>

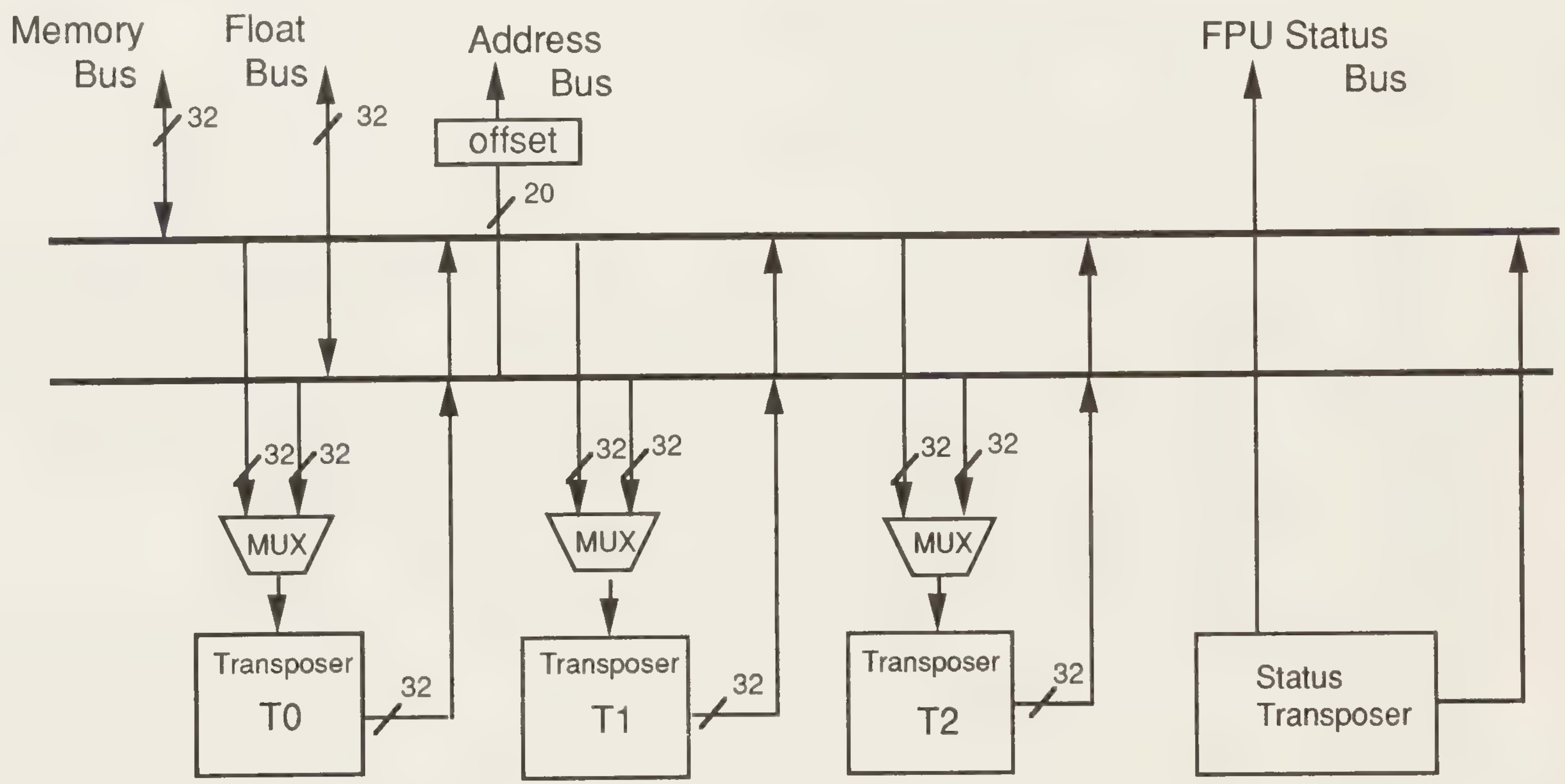
- 8MHz bit slice controller
- 128 bits x 64K micro control store
- Synchronous with CM

Processor Section



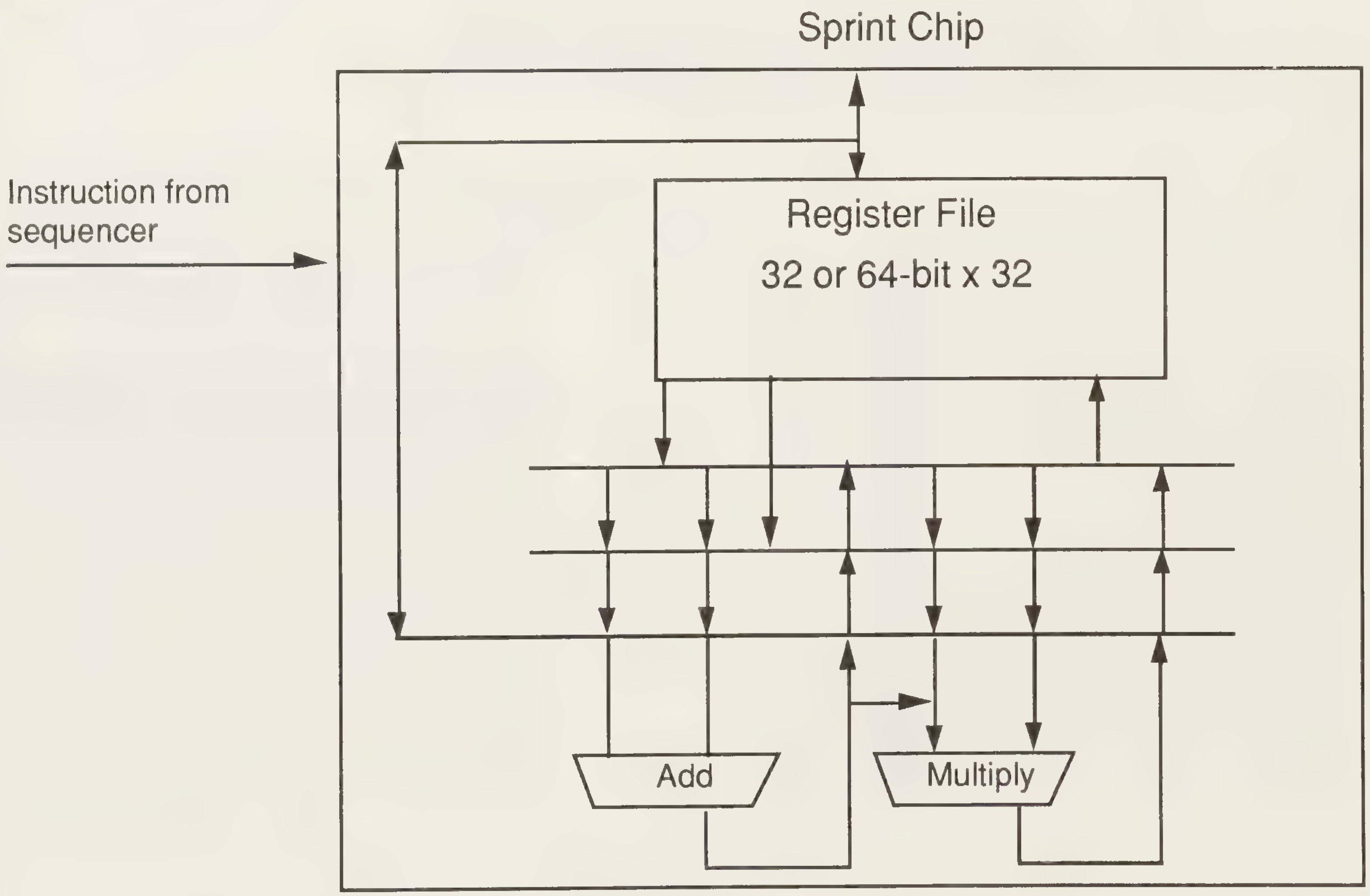
- Section is repeated 2,048 times in a full CM-2
- Instructions are broadcast
- Memory addresses can be broadcast or locally generated (indirect addressing)
- The memory and floating point units are commercial parts
- CM and Sprint chips are standard cell/custom CMOS parts

Sprint Chip



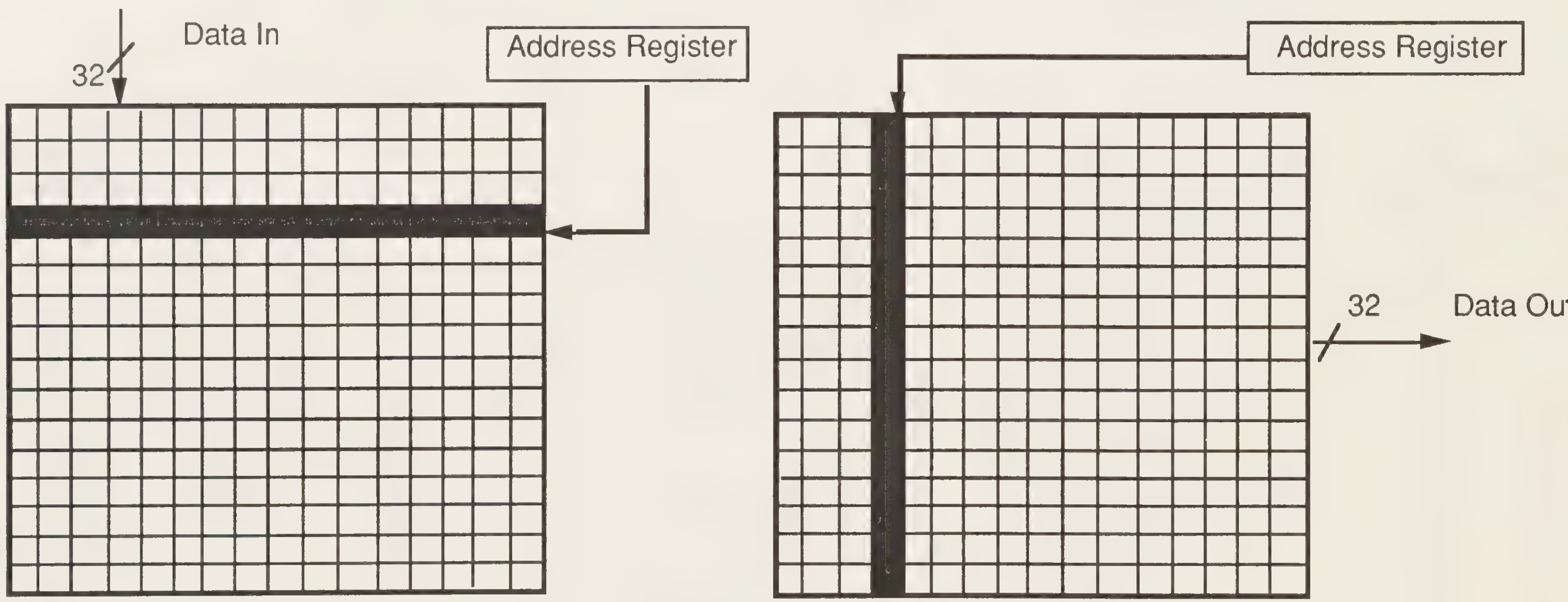
- Bit serial ⟷ word parallel conversion
- Floating point interface
- Indirect addressing

Floating Point Unit



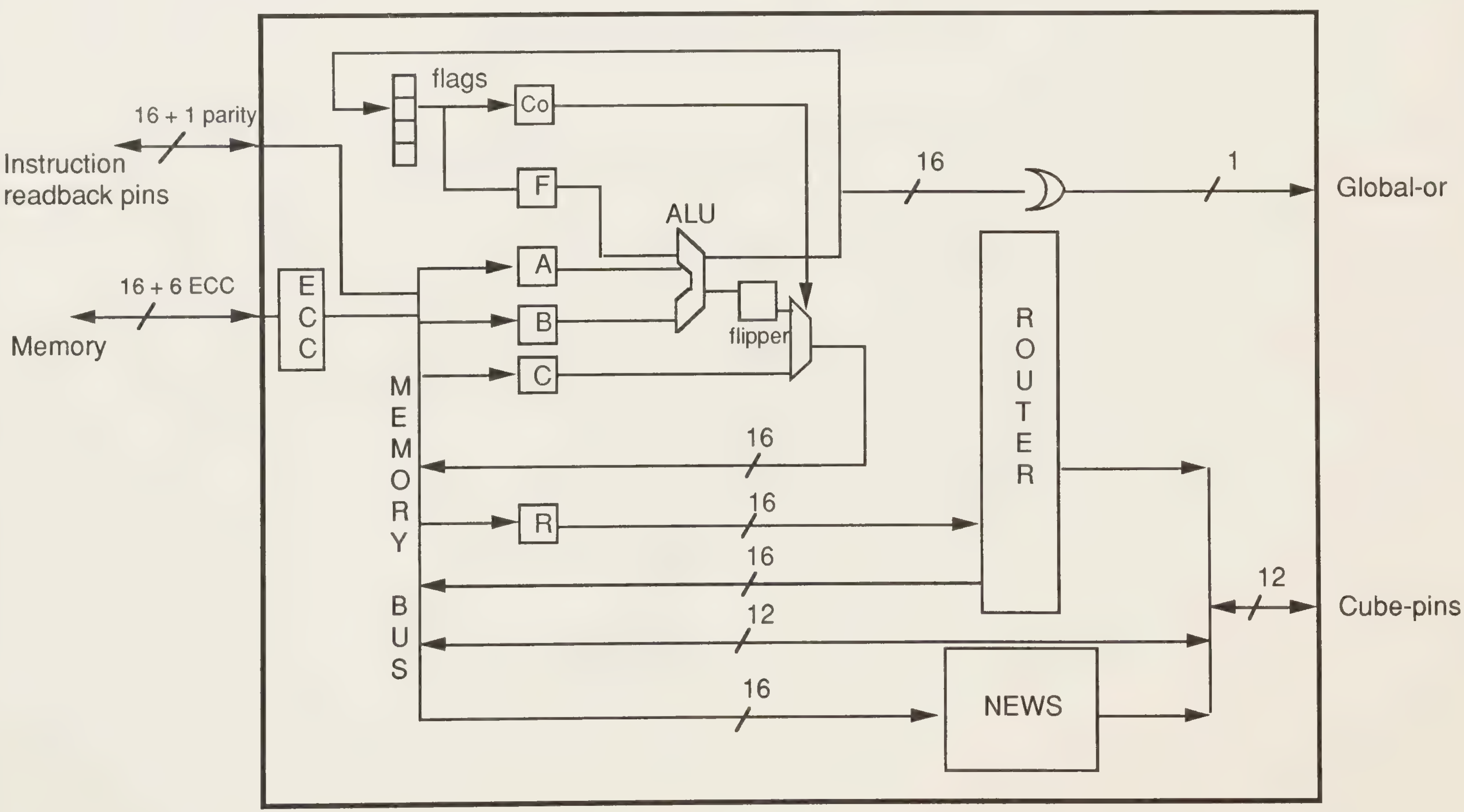
- Standard IEEE part
- Batch 32 processors gets vector performance
- Virtual Processor pipelining
- Floating point add or multiply is one "Add times"
- Floating point divide is 3 "Add times"

Transposer Diagram



- bit serial → word parallel conversion
- word parallel → bit serial conversion
- 32-cycle latency

Connection Machine Chip



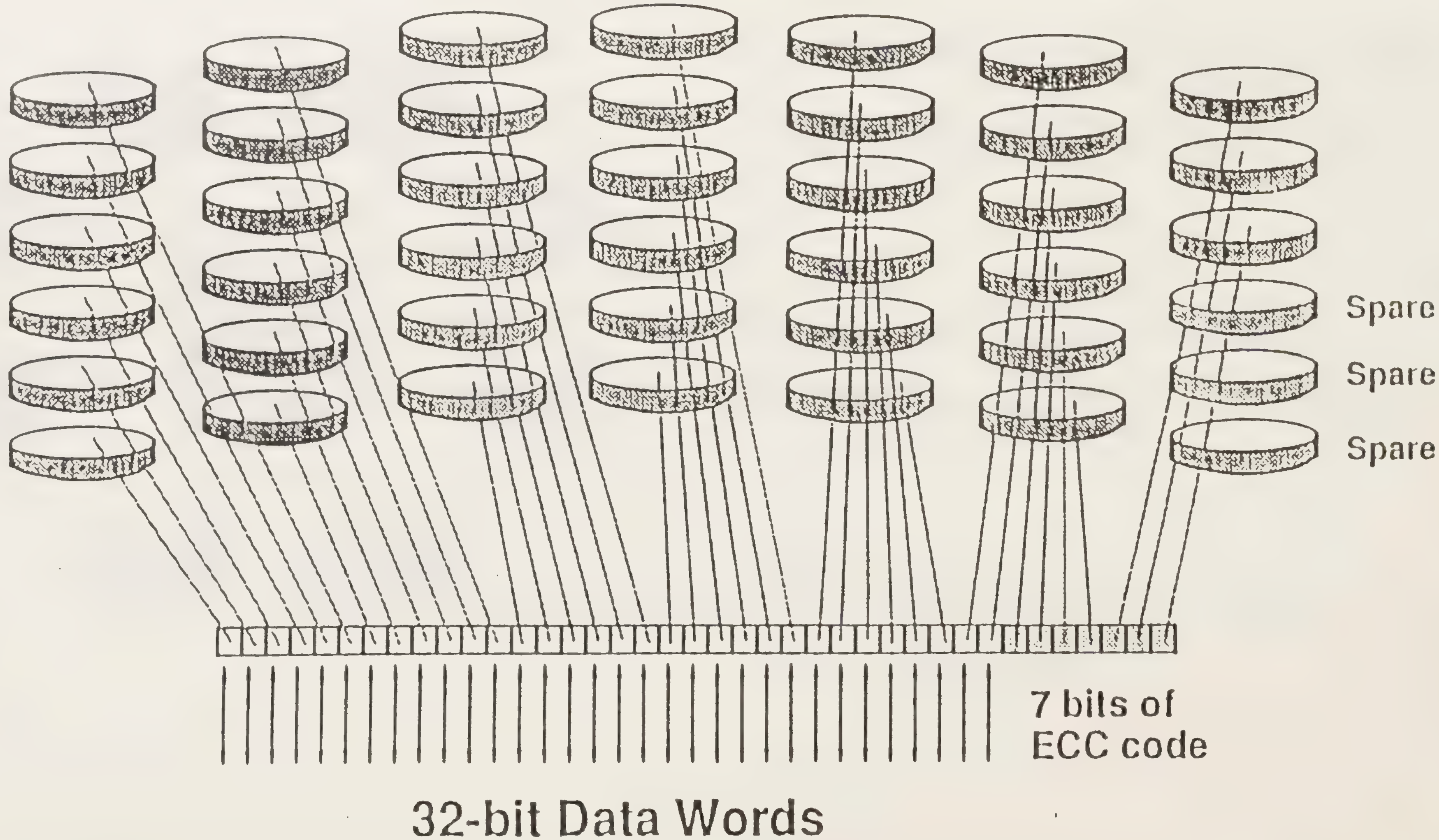
DataVault Hardware

- Parallel File System
  - 32 + 8 ECC + 3 spare drives
- High Data Transfer Rate
  - 25 - 40 MBytes per second
- Data Protection
  - Full ECC

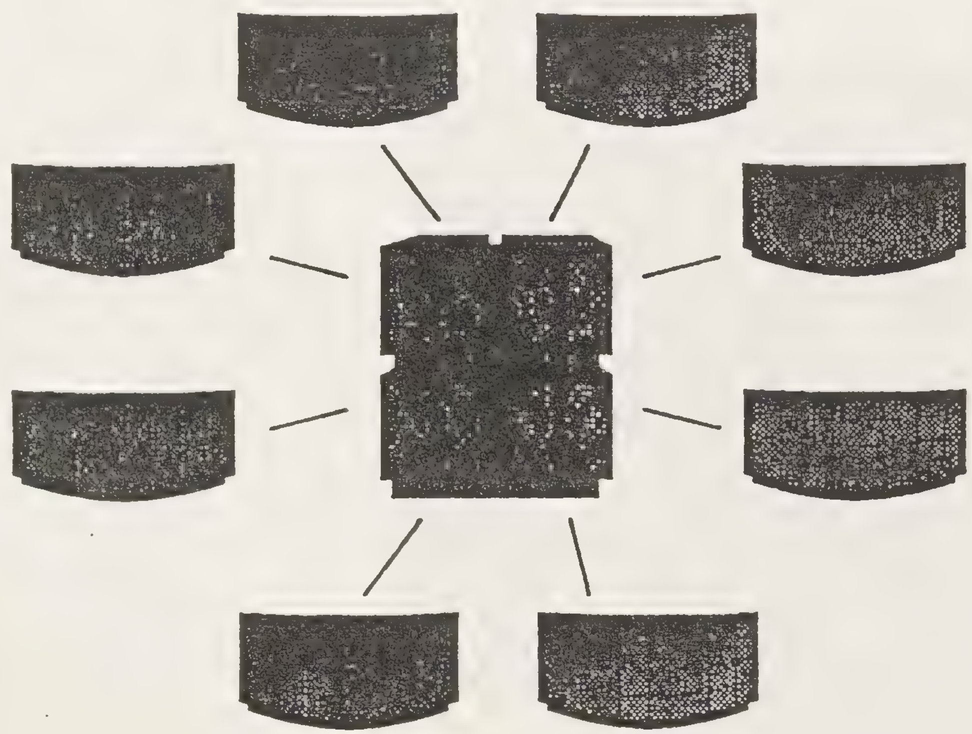
I/O SYSTEMS

- UNIX - LIKE HIERARCHICAL FILE SYSTEM
- EACH DEVICE IS INDEPENDENT FILE SYSTEM
- EACH DEVICE HAS A FILE SERVER
- FRONT END AND ALL FILE SERVERS ARE LINKED VIA LOCAL ETHERNET FOR COMMAND AND CONTROL
- STANDARD UNIX DEVICE / FILE SYSTEM CALLS

Parallel Data Transfer

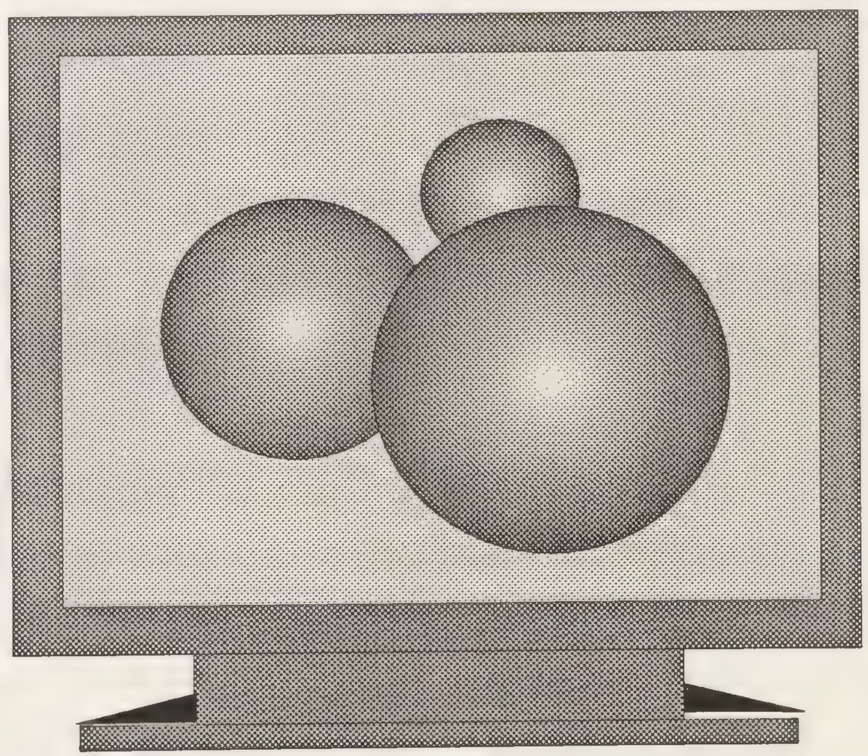


DataVault Mass Storage



- > 150 MBytes per second aggregate transfer rate
- Up to 64 DataVaults (2.5 TBytes) per system
- Full redundancy and Data Healing

Color Graphic Display



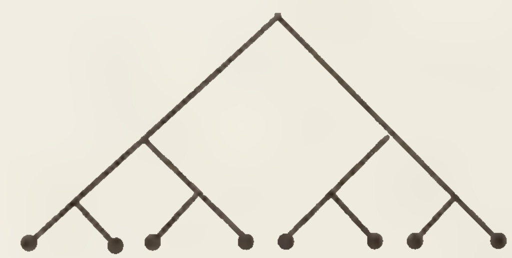
1280 x 1024

Real Time Speed

Frame Buffer Hardware

- Pixel resolution
    - high: 1280 x 1024 60 Hz non-interlaced
    - NTSC (television): 640 x 480 30 Hz interlaced
- Color resolution
    - "24 bit" direct color (red, green, blue: 8 bits each)
    - "8 bit" pseudo-color (uses color lookup table)
    - overlay (4 bits)
- High speed
    - plugs into Connection Machine backplane
    - time to write one full image on a 16K CM-2
        - NTSC 8 bit: 15 to 50 ms
        - high 24 bit: 165 to 270 ms
- Panning and zooming

SCANS

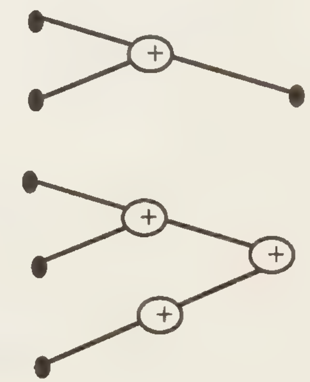


- Use all hypercube wires directly
- Very fast operation because of the regularity
- Performance: 25 "Add times"
- Faster with high virtual processor ratio

SCANS: An Example

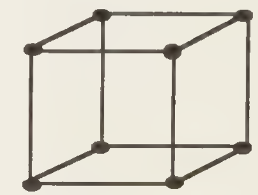
Data	5	3	4	2	50	30	24
Max - Scan (inclusive)	5	5	5	5	50	50	50
Add - Scan (inclusive)	5	8	12	14	64	94	118
Add - Scan (exclusive)	0	5	8	12	14	64	94

In-Router Combining



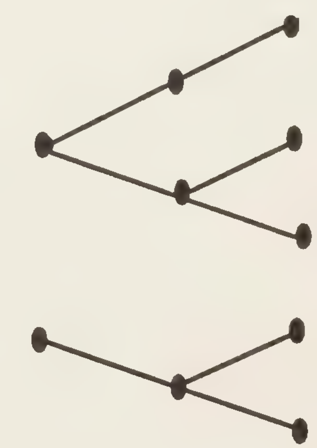
- Improves performance for clustering sends, e.g., histogramming
- Run time fan-in tree construction
- Detects message with same destination on each router and combines them
- Operations
  - Integer Add
  - Integer Max
  - Overwrite
  - Logical Or

Router



- One router per CM chip, 4096 (212) total
- Each connected to 12 neighbors along hypercube
- Packetized, pipelined, load balancing, store and forward algorithm
- In router combining
- Store router state
- Backwards routing
- Sprint routing
- Performance
  - 1.5 miliseconds per VP for 32-bit random send
  - ~1 GBit per second throughput\
  - 75 "Add times"

Backward Routing (Hardware "Get")

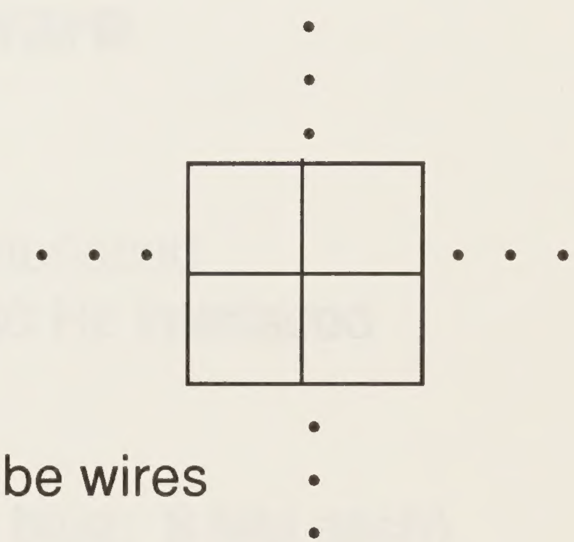


- Eliminates hot-spots on "shared-memory" accesses
- Router can store its switch state in memory, including combining information (~ 400 bits per virtual processor)
- Router can run "backwards"
- Run time fan-out tree construction
- Routing time is 2 times forward routing

Sprint Chip Routing

- Two tier routing
  - hypercube routing
  - indirect addressing
- Indirect addressing for message delivery
- Linear routing time with virtual processors
- Part of Release 5.0

NEWS



- 1, 2, 3, 4, . . . dimensions supported
- Grid communications over the hypercube wires
- Faster than the router
  - no address part of the message
  - local messages get delivered directly
- Performance: 6 "Add times"
- Faster with high virtual processor ratio

Frame Buffer Hardware

- Pixel resolution
  - high: 1280 x 1024 60 Hz non-interlaced
  - NTSC (television): 640 x 480 30 Hz interlaced
- Color resolution
  - "24 bit" direct color (red, green, blue: 8 bits each)
  - "8 bit" pseudo-color (uses color lookup table)
  - overlay (4 bits)
- High speed
  - plugs into Connection Machine backplane
  - time to write one full image on a 16K CM-2
    - NTSC 8 bit: 15 to 50 ms
    - high 24 bit: 165 to 270 ms
- Panning and zooming